ISSN 0265-2919

R 80p

THE HOME COMPUTER COURSE 5

MASTERING YOUR HOME COMPUTER IN 24 WEEKS

An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

Next Week

- **We feature the Atari 400 and 800. Their colour graphics have encouraged software writers to create some amazing games**
- **THE HOME COMPUTER COURSE looks at the exciting changes the computer will make in your own home**

Editor Gareth Jefferson; **Art Editor** David Whelan; **Production Editor** Catherine Cardwell; **Picture Editor** Claudia Zeff; **Designer** Hazel Bennington; **Art Assistants** Steve Broadhurst, Julie-Anne Chambers, Liz Dixon; **Sub Editor** Teresa Bushell; **Researcher** Melanie Davis; **Consultant Editor** David Tebbutt; **Project Manager** Jimmy Egerton; **Contributors** Tim Heath, Peter Jackson, Ray Hammond, Ron Smith, Ian White; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Published by Orbis Publishing Ltd; © 1983 by Orbis Publishing Ltd; Typeset by Universe; Reproduction by Muills Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

How to obtain your copies of HOME COMPUTER COURSE – Copies are obtainable by placing a regular order at your newsagent.

Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER COURSE Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

How to obtain binders for HOME COMPUTER COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 4, 5 and 6. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595 Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency In case of difficulty write to HOME COMPUTER COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

School On Screen

IAN DOBBIE/TAKEN AT THORNHILL PRIMARY SCHOOL

Computers In Primary Schools
Britain is preparing for the future with an educational policy to introduce computers to children in their first years at school. The idea is not only to familiarise school children with new technology, but to use the computer to teach a wide range of subjects — from biology to foreign languages. It is an ideal and patient teacher as it only moves on to new ground once a problem has been mastered, allowing both slow and fast learners to work at their own pace. The computer has a wider use as a learning aid — simply by using it the child discovers how a problem is analysed and solved

The classroom has now caught up with the computer age and an exciting choice of educational programs is on offer

Every one of Britain's 29,000 primary schools will soon have a computer and many secondary schools already possess one. Nowadays, computers are not simply on the curriculum under 'Computer Studies', but are being used by teachers for teaching numeracy and literacy, helping slow-learning children, and teaching foreign languages.

There are many educational programs for home computers on the market, but teachers frequently complain about their poor quality. The reason for this is that few programs have been written that observe both educational and computing disciplines.

Computer programmers rarely have teaching experience, and teachers, many of whom are new to programming, have sometimes been responsible for the most rudimentary of programming mistakes.

Although a teacher's program is likely to work in his or her own classroom, the moment it is sent out to another school problems arise. The actual program, whether stored on cassette or disk, is usually not sufficient by itself: good explanatory documentation should also be considered equally essential.

Without this, students may be unable to operate the program. 'Well of *course* you have to type LOAD!' might be the programmer's response to the problem, but to a computer novice all such details must be spelled out.

More seriously, good programming calls for an anticipation of all the mistakes that a beginner might make. This is as important as ensuring the program is a successful teaching aid. Good programming means more than 'de-bugging' the program to the point at which it does what it should when the right key is pressed. It also means ensuring that the program doesn't do anything it shouldn't when the wrong key is pressed. This is the hardest part of program writing. The program must be able to recover from the most careless of errors by a child, and still leave him thinking the computer is easy and fun to use.

Despite these problems, there is a wide range of educational programs suitable for home and school use. Computers are wonderful educational tools, and in choosing software for your children it is useful to understand the different ways in which a computer can be used.

A computer can be used to instruct a child in almost any subject. If the program is good, the child is likely to be fascinated by it and motivated to learn.

The usual type of educational program is best described as 'drill and practice'. Children are

shown examples and then asked to solve similar problems. Usually the program keeps score of how well the pupil does. It even offers encouragement when the pupil gets the answer right, and makes a gentle suggestion to 'try again' when the answer is wrong.

Deciding which programs are right for your child depends on several things: the age of your child, the make of your computer and what your child is studying at school.

If you have yet to buy your home computer but you suspect that education will be one of its important roles in the home, it is worth finding out which type of computer is being used in your child's school. If you are able to purchase a similar model, the education programs your child is using at school will be available to you at home. Many schools are delighted to offer parents copies of the programs they are using in class and this 'homework' can have a considerable benefit. If you have already bought your home computer and it is not compatible with the school's computers, don't despair; there is value in the very variety of computing experience your child is receiving.

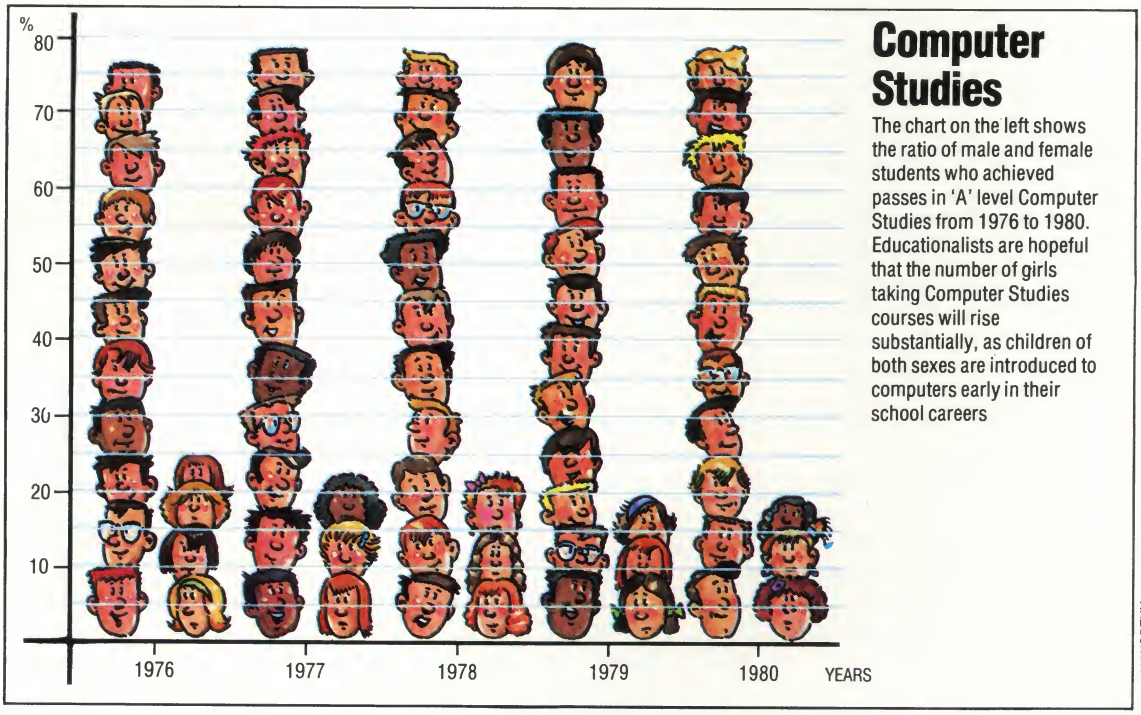
It is natural that more educational programs are available for the better established computers, but some manufacturers have placed a particular emphasis on education. There are particularly wide ranges of educational programs available for the Apple, the Commodore PET, the Tandy, the BBC Micro, Sinclair and Texas Instruments but some of the newer manufacturers have yet to attract a really wide range of programs. Educational programs for any computer are likely to be available from several sources. The manufacturer of the computer is one, and the various independent software houses is another. The latter write programs for computers and advertise in the computer magazines and home computer shops.

The Right Choice

For the under eights most programs concentrate on the basic skills of literacy and numeracy. One of the attractive ranges of educational software for very young children is produced by Texas Instruments. The TI-99/4A Home Computer has been slow to catch on in Britain, but many parents have been impressed by the range of TI educational software produced by both TI and Scott, Foresman & Co. in the USA. The 99/4A is actually a 16-bit computer and this means that programs written in machine code are likely to be very much better than programs written on the more usual 8-bit home computers.

This is proved by such TI programs as 'Beginning Grammar', 'Addition and Subtraction' (1 and 2) and 'Number Magic'. These programs are stored on a plastic cartridge, which slots into the TI 99/4A and is easy for very young children to use. And if you've splashed out on the marvellous little voice synthesiser add-on for the 99/4A, you will know that several of the programs talk in a Dalek-type voice children love. The problem is that the programs are American. The odd word is spelled differently and some British teachers would be likely to go into paroxysms over some of the TI programs. However TI has a stunning example of LOGO available (see page 34), although this really falls into the second category of discovery tools.

There is a good choice of programs available for home computers. A comprehensive selection can be found in *Educational Computing*. Here you will also find the program houses advertising their



Computer Studies

The chart on the left shows the ratio of male and female students who achieved passes in 'A' level Computer Studies from 1976 to 1980. Educationalists are hopeful that the number of girls taking Computer Studies courses will rise substantially, as children of both sexes are introduced to computers early in their school careers

ANNI AXWORTHY

wares to teachers and you will find informed reviews of the various programs. The sort of programs for young children available for the BBC Microcomputer include the usual 'Number Fun' type and many basic literacy programs. Many useful programs are available from the British Micros In Primary Schools organisation. One is 'Cat and Mouse', which helps the child become familiar with the keyboard layout of the BBC Microcomputers.

As children get older so the number and range of programs increase. Programs for the 8 to 11 age group vary in complexity and quality, and most concentrate on reinforcing basic skills and stretching the child's ability. This age group acquires special interests such as music and foreign languages, which can be taught by computer. Most computers have this type of program available.

In the secondary school age group there is a plethora of programs. The only way to wade through them and to pick out the best is to speak to your child's teacher. It is important that your child's home study is not in contradiction to the work he is getting at school and most teachers will prove very helpful in guiding you towards the right type of program.

A further category of educational computer programs is primarily concerned with children under 13. At this age children are still discovering how to learn, and programs which induce them to use the computer to discover the world for themselves must prove very valuable. The best known program is LOGO, and a version of this language is available for computers made by Atari, Tandy, Apple, Texas Instruments, Research Machines, Commodore and IBM. Versions are promised for the BBC and for Sinclairs, but they have yet to materialise. Using this program a child, between the age of 6 and 12, is encouraged to explore the computer's drawing power (and, in turn, geometry) with a 'turtle'. The child discovers how to teach the turtle to remember procedures (programs) and on some implementations the child progresses until he or she can draw a fantasy world of 'sprites' on the screen. Using this program children are actually teaching themselves the basic laws of mathematics, and much has been claimed for the power of this program to teach mathematical and spatial concepts.

It is not easy to choose good educational software, because there is so much to choose from. It is a good idea to attend one of the many computer exhibitions that are periodically held around the country. Here you will be likely to find both manufacturers and program writers showing off their wares, and you will have a limited opportunity to see, try and compare programs. The shortage of good programs that satisfy both educational and computing requirements is not likely to last for long. Each month more programs emerge that are likely to provide a valuable stimulus to your child's development.

The Secret Agent

There are many educational programs now on the market. This one is called Secret Agent and is published by Heinemann. Your mission is to catch a notorious enemy spy on the loose in Europe before he eliminates all your agents. The clues to his whereabouts have to be deciphered before you can catch up with him. The chase is on...

The master spy may only be apprehended in a city – but he won't stay in one place for more than a couple of hours. Once you think you know where he is, you have a choice of travelling there either by train or aeroplane. You have to decide for yourself whether speed is more important than cost

A light will flash on the map whenever a message is sent from one of your agents, showing the city from which it is being sent. If your agent is eliminated, the message will be intercepted before you can receive it. You may then wish to hire a new agent, but you will have to pay for it

Informers are happy to help you by selling their reports, but these will be sent in code so you may need to appeal to the boffins in London for help. At the end of the game you will have learnt the names and locations of all the European cities, and developed an understanding of timetables and the knack of careful budgeting

IAN McKINNELL

Cracking The Code

Type your computer language onto the keyboard and a single program working within the micro will speedily convert it into the machine's own code

Although microcomputers appear to perform similar functions, each model is unique. Some are supplied with programs already built in, while others require such programs to be 'read' in from an external disk or cassette tape.

Some machines contain a single, all-embracing program that allows both the entry of programs, and the use of direct instructions such as SAVE or LOAD. Other models need separate programs to carry out these functions.

There are, however, similar principles upon which most popular microcomputers operate. The movement of information to and from external storage devices (disks and tapes) to the screen is, in each case, controlled by the keyboard. Also, every machine can communicate with other external devices such as printers, plotters, and scientific instruments. And most micros allow their users to write programs in languages similar to English, such as BASIC for example.

When you type a BASIC program at the keyboard of your computer, a program called the 'operating system' passes what you have typed both to the screen, and to a BASIC interpreter program. This means that three programs are being held inside the computer at the same time; namely the operating system, the BASIC interpreter and your own program.

When you run your program, all three programs would appear to be active at the same time. Each BASIC instruction in your program is translated by the interpreter and, one by one, the resulting machine code instructions are passed to the microprocessor for action. At the same time, the operating system is checking the keyboard for data entry and possibly displaying this on the screen.

If one of your program instructions asks for something to be printed or written on disk, for example, then the interpreter would request the operating system to carry out this task.

The illusion that several things are happening at once is due to the microprocessor's incredible speed. It can process instructions from the operating system and the interpreter so quickly that they can both be served at the same time.

Some machines can run even more quickly by allowing the arrival of data to 'interrupt' normal processing. In this way, there is no need for the operating system to check the various external sources of data, such as the keyboard or disk drives.

A less sophisticated type of word processor is called an 'editor'. These tend to vary considerably in quality, and you will probably find the editor built into the BASIC interpreter equally as good.

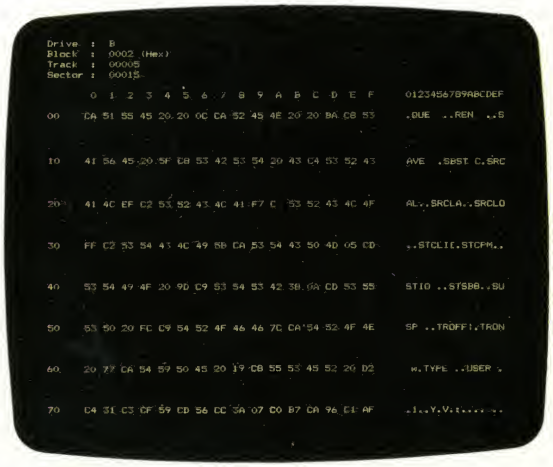
Instead of using an interpreter to run your BASIC programs, you could use a compiler. Whereas the interpretor has to translate each instruction every time it is encountered, the compiler translates your entire program into computer machine code for once and for all. Programs which have been 'compiled' run much more quickly than their 'interpreted' counterparts.

BASIC is a very popular language for writing programs. It has the advantage of being very close to plain English, and is ideal for beginners. But the more adventurous programmer can make his programs run much faster by using an assembly language. This is not like English at all, therefore the programmer must have a fairly detailed knowledge of how the microprocessor performs its functions.

Each instruction that you give the computer has a direct equivalent in machine code. An assembly language is a collection of abbreviated instructions such as MVI (Move Immediate) or JZ (Jump on Zero). These are used to help the programmer remember their functions.

If you master an assembly language, then the next thing to tackle would be machine code, but there would be very little point unless you really needed to shave tiny fractions of a second off a program's execution time.

Machine code on microcomputers is generally written in a form called Hexadecimal. This is a form of numbering to a base of 16. You count from 0 to 9 normally, then go on to use the letters



Disk Operating System
When programs are stored on a floppy disk, the information is distributed at random around the surface of the disk. The disk operating system is a program that automatically keeps track of the location of each byte of information.

The illustration shows the information stored on one small section of a disk. This is represented in 'hexadecimal' form in the left-hand block and the equivalent character is shown on the right. Codes that do not correspond to characters that are to be printed are shown as dots

A to F for the numbers 10 to 15. Each memory location comprises eight binary digits (bits) and these may be represented by a pair of 'hex' digits.

For example, the binary pattern 01011101 would first be split into two halves: 0101 and 1101. These would be translated into the decimal numbers 5 and 13 — which equate to 5 and D in 'hex'. Thus 01011101 is referred to as 5D when programming in machine code. This is the slowest way to develop programs, but probably produces them with the fastest execution times. The following are some extracts from typical programs:

```
BASIC
100 INPUT "Enter hours worked"; HOURS
200 PAY = HOURS * RATE
```

The first line displays a message on the screen inviting the user to enter the hours worked. It accepts the input, then in the second line, multiplies it by a rate of pay (entered earlier in the program) to give a gross pay figure.

```
Assembly Language
MVL  C, 01
CALL 05
```

The first instruction above moves the value '1' into a part of the microprocessor's memory called the 'C register'. The second instruction asks the operating system to take control. The operating system then passes control back to your program. Here is a direct translation of the assembler lines above:

```
Machine Code
0E01
CD0500
```

A machine code translation of the two BASIC statements would comprise many commands. It is clearly preferable to allow a compiler or an interpreter to do this work rather than to write in assembly language or machine code.

Some operating system commands are almost cryptic as assembly language programs. Here are some examples from CP/M (Control Program for Microcomputers):

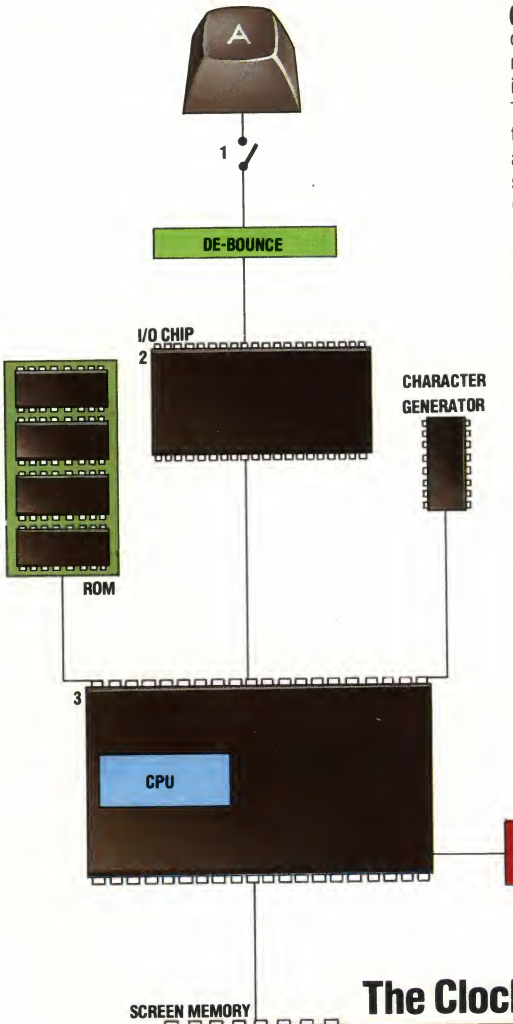
```
DIR *. BAS
```

The above means 'list all the files on the current disk drive whose suffix is BAS'. DIR is an abbreviation of Directory.

Despite this rather odd approach to its commands, CP/M is installed on more than a million machines. It has massive advantages for professional software writers. Programs can be easily transported from one machine to another, provided they are written in such a way that they pass control to CP/M to handle the disks, keyboard, printer, and the screen.

Manufacturers who introduce a new machine can benefit from an existing base of software, and would-be purchasers can buy machines, confident that their software requirements are likely to be met.

From Key To Screen

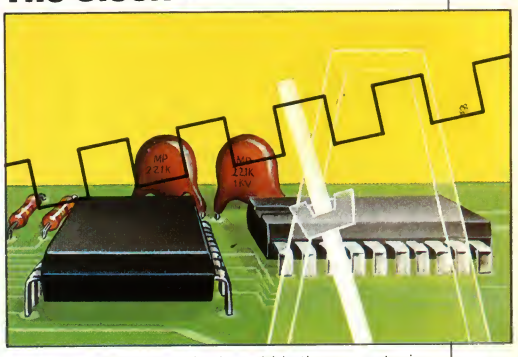


(1) Pressing a key on a computer keyboard usually results in a letter appearing immediately on the monitor. The process is more complex than it looks, however. When a key is pressed, a signal is sent via the I/O chip **(2)** to the CPU. The CPU **(3)** runs a small program in the ROM that works out which key has been hit, and puts a corresponding code in the memory that is currently being used as storage for the screen image, screen RAM **(4)**.

Meanwhile, the Video Circuitry **(5)** is constantly scanning the screen RAM, and by fetching the bit patterns associated with each letter from a 'character-generator' ROM, sends a signal to the television or monitor, which then draws the letter on the screen **(6)**, in the right place, and if selected, in one of the available colours.

All this apparently frantic activity happens in a few millionths of a second

The Clock



Timing within the computer is critical. For example, if the CPU reads a byte from memory, it must set the address lines to the byte location. The CPU needs to be sure that the memory chip has had time to put the byte of information onto the data bus. Only then will it accept and 'read' the byte into the CPU's internal memory. The clocks in home microcomputers 'tick' as fast as four million times a second

STEVE CROSS

From Abacus To Apple

The modern microchip owes its existence to the genius of inventors whose work spans three centuries

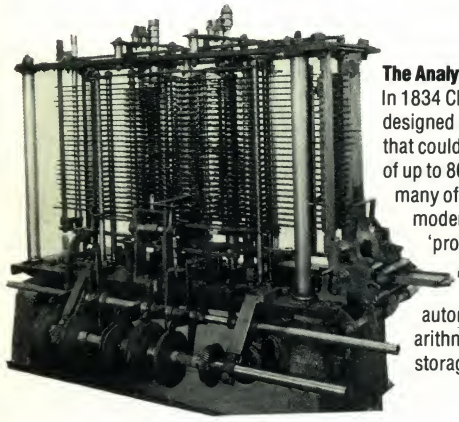
When you're in love you think you can do anything — at least that's what Charles Babbage thought in 1830. As a result he almost built the world's first working computer 100 years before its time.

There were several drawbacks in Babbage's design: one was the fact that the computer had to be mechanical; and another was the shortcomings of engineering at the time. But despite these problems, Charles Babbage built a machine that so impressed the government of his day that they promptly gave him a grant of £1,500, a sum which later grew to £17,000. (Equivalent to £1,700,000 today.)

The story of computing, however, begins much earlier. A computer is a machine that can be told how to compute a group of numbers, will remember what it has computed and can be adapted to compute another set of numbers. The earliest example was the abacus, in use in 2,000 BC and still found today in Japan and Eastern Europe. It is a very useful type of computer because you can physically see the sum on the wires — the position of the beads forms a 'memory' of the sum. But they're not automatic, and they're not useful for large numbers.

Blaise Pascal, a Frenchman, invented the world's first mechanical calculator in 1642 — possibly to please his father who was the local tax inspector. It worked perfectly: carrying numbers from the 'units' column to the 'tens' column by a trip device, in much the same way as a car speedometer carries numbers, and it was totally functional. Modestly, Blaise called it a 'Pascaline'.

Although the Pascaline did not sell well, it sparked off great scientific interest and over the next few years many improvements on the first calculator were made. Nothing of significance emerged, however, until Charles Babbage and Ada Lovelace started to think about the problem.



The Analytical Engine
In 1834 Charles Babbage designed an Analytical Engine that could handle computations of up to 80 digits. This included many of the features of the modern computer. The 'programs' were controlled from punched cards and the results were printed automatically. It also had an arithmetical 'mill' and separate storage devices

COURTESY OF SCIENCE MUSEUM

Blaise Pascal
The Pascaline was the world's first mechanical calculator, designed by the Frenchman Blaise Pascal in 1642. He originally wanted to build a machine that could divide and multiply, as well as just add and subtract. The Pascaline used a stylus to move the wheels, and had a special mechanism that carried digits from one column to the next. Pascal was granted a patent by the king of France so he could market his calculator, but it was never a financial success

Countdown To Computing

1000 BC
ABACUS
Still in use today, it calculates and stores numbers using beads

1614
LOGARITHMS
John Napier invented this aid that reduced division to subtraction and multiplication to addition

1623
BINARY
Francis Bacon first used base 2 arithmetic

1642
ADDING MACHINE
Blaise Pascal invented his mechanical 'Pascaline' calculator

1671
CALCULATOR
Gottfried von Leibnitz succeeded in making a machine that could multiply and divide

1802
PUNCHED CARDS
Joseph Jacquard built a loom that stored the pattern of the fabric on punched cards

1822
DIFFERENCE ENGINE
Charles Babbage's first mathematical calculator

1000 BC // 1614 1623 1642 1671 1802 1822

Charles Babbage was born into a wealthy family in 1791. Despite a comfortable upbringing, he proved to be a mathematical genius and because he grew frustrated at amending the many mistakes he found in logarithm tables, he turned his mind to building a machine that could take the drudgery out of calculations.

In 1822 he showed the Royal Astronomical Society his first model of a 'difference' engine, a machine that could make the calculations needed for constructing logarithm tables. The name derives from an abstract mathematical technique known as the method of differences. The society encouraged him to go onto further and better machines.

Together with Ada Lovelace, the daughter of Lord Byron, he set out on a more ambitious project to build an 'analytical engine'. This machine was designed to calculate values of mathematical functions that were far more complicated than the logarithmic functions.

This machine was fraught with problems from the very start. It just wouldn't work. The drawings that have survived show us that the construction was huge, filling the large workshop Babbage had built on his estate. The hundreds of cogs, rods and wheels had to be specially turned on lathes and current metal technology simply wasn't good enough. When he had built his little model, the minor inaccuracies it produced could be shrugged off, but once he tried to get the full-sized machine going, the minor inaccuracies became greatly magnified.

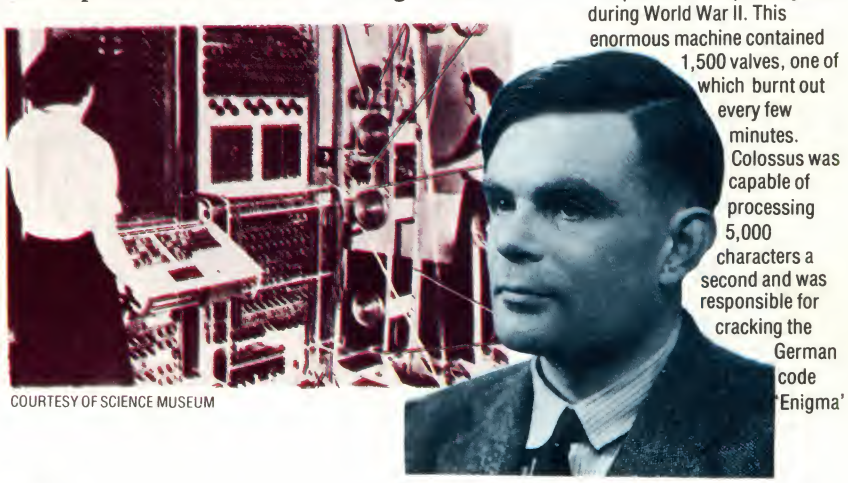
Babbage was on the right track, and had he been able to get parts machined sufficiently well, it is probable that his analytical machine would have worked. Much of the logical architecture and design structure of today's computers can be traced back to Charles Babbage and he is remembered as one of the founding fathers of modern computing.

One important hunch that occurred to Babbage during his years of work was the idea that his engine could be 'programmed' or 'taught' to do any mathematical task. Had he been able to prove this, or had he been able to build a machine that could do it, the Victorians would have been running their Empire by steam computer.

It wasn't until 1936 that proof was provided for Babbage's hunch. It appeared in an obscure paper called *On Computable Numbers,* published by a young Cambridge mathematician, Alan Turing. Turing's name may be almost unknown to the general public, but his contribution is fundamental to the development of the ideas that had to be generated before the computer could become a reality. Scientists had for a long time reasoned that mathematics was not a mysterious art but a science totally controlled by logical rules and that if you gave a machine these rules and a problem, it should be able to solve it. However, all the efforts of the most able mathematicians had failed to develop such a machine. Turing decided to

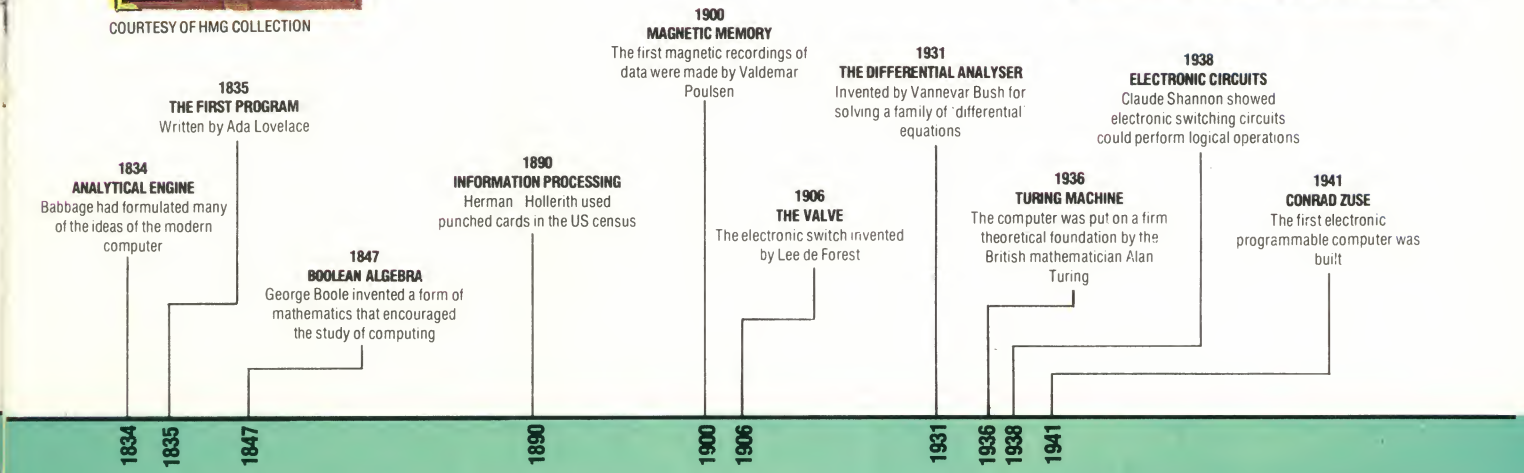
The First Programmer
Countess Ada Lovelace, Charles Babbage's companion and Lord Byron's only legitimate daughter, is one of the few women to figure in computing history. A gifted mathematician, she understood Babbage's Analytical Engine and wrote some of the best accounts of how it worked. She even devised programs for it, making her the world's first computer programmer

COURTESY OF HMG COLLECTION



COURTESY OF SCIENCE MUSEUM

Alan Turing And Colossus
Alan Turing proved that a set of simple instructions could solve any complex problem. He and his team developed Colossus, one of the world's first computers, seen operating here during World War II. This enormous machine contained 1,500 valves, one of which burnt out every few minutes. Colossus was capable of processing 5,000 characters a second and was responsible for cracking the German code 'Enigma'



approach the problem in a different way. He looked at the type of problem that a machine following logical rules could solve and tried to list them all. If they comprised the whole of mathematics then the conjecture would be solved.

Turing led a research team in Buckinghamshire and developed the most secret invention of the Second World War, Colossus, the world's first electro-mechanical computer. It was this machine that cracked the German 'Enigma' message codes throughout the war.

After the war, Turing went to America to lend his genius to the USA's first computer project. With his help, the first American computer was built. This computer was called the ENIAC and was developed at the University of Pennsylvania. It used 18,000 valves; one of these would blow every two minutes!

One of the reasons why Turing's name is virtually unknown is because he was working for MI6 and enveloped himself and his work in secrecy. The British government did not release details of Turing's pioneering work until 1975.

Computer development surged forward but it wasn't until the invention of the silicon transistor in 1947 that rapid computing became possible.

Transistors can do everything a valve can do but they do it faster, more reliably and without generating heat. Like valves, they are electronic switches that can be switched on and off and can be used to represent either the zeros or the ones of the binary code. Throughout the 50's and early 60's larger and faster computers were built and they were used by big business as well as governments.

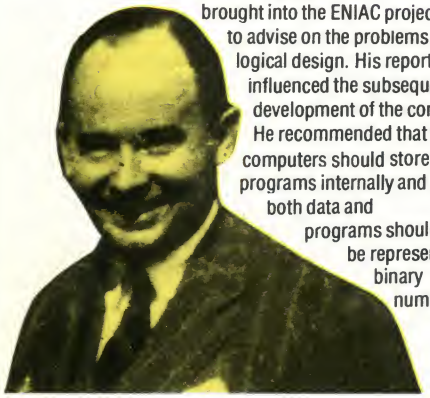
In the mid-60's, scientists reasoned that an electronic circuit would work just as well if it were miniaturised. With billions of space race dollars behind them, laboratories started experimenting in placing circuit designs on a single chip of silicon and then etching the design onto the chip. Before the end of the 60's this 'integrated circuit' was born and computing had taken a massive leap forward.

The development of a 'circuit-on-a-chip' led naturally to a 'multiple-circuit-on-a-chip' and the microprocessor was the inevitable outcome of layering several chips together.

Although microchip technology bears little resemblance to the giant analytical engine that Babbage and Ada Lovelace built, and scarcely more to Turing's Colossus, the practical 'architecture' that Babbage created is still used in today's microprocessor. The theory that made it possible, Turing's mathematical proof of the feasibility of computing, is unsurpassed.

The 'Architect' Of The Modern Computer

John Von Neumann was brought into the ENIAC project to advise on the problems of logical design. His report influenced the subsequent development of the computer. He recommended that future computers should store their programs internally and that both data and programs should be represented by binary numbers



COURTESY OF LONDON MATHEMATICAL SOCIETY

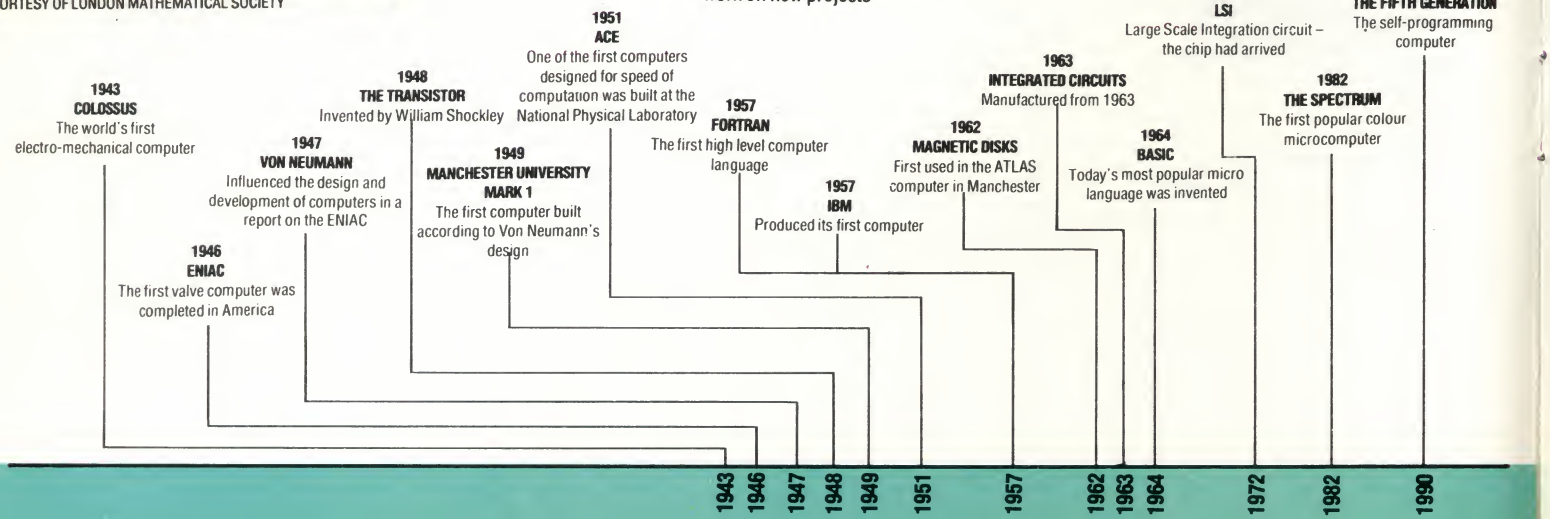
Steve Wozniac

Steve Wozniac is often referred to as a 'living legend' in the computer business. Pictured here with his first invention, the Apple I (now a collector's item) Wozniac designed computers at school. Although he never trained as an engineer, he did more to miniaturise the computer for cheap home use than anyone else. The Apple II, which he developed in 1976, had the first home computer disk drive and is still one of the world's biggest sellers. Wozniac has recently returned to the Apple Corporation to work on new projects

The Sinclair Spectrum

Developed less than 40 years after those first cumbersome computers that were housed in whole rooms, the Spectrum is small, compact and cheap. The Spectrum was the first personal computer with colour capability that you could buy for less than £100. The first computers were huge complex machines, built by governments and

institutions, and operated behind closed doors. Nowadays, thanks to machines like the Spectrum, most families can afford a home computer



BBC Model B

A fine technical specification backed by the marketing and programming resources of the BBC has made this micro popular in both schools and homes

The BBC Model B is manufactured by Acorn Computers in Cambridge and marketed by the BBC. There were originally two models, but the cheaper and less sophisticated Model A is now being phased out.

The Model B has sold very well to schools and is on the official Government approval list for this purpose. A considerable amount of educational software has been developed for this machine — from programming languages to Computer Aided Learning packages.

The technical specification of the Model B is still considered to be extremely fine, despite the number of new machines that have been launched since its introduction. In particular, the programming language BBC BASIC is very well equipped with commands to cope with the special functions. It also makes the task of developing and editing programs easier.

There are eight distinct graphics modes. This means that the user has a choice of low, medium or high resolution, though in the latter case the number of colours available is limited. The maximum resolution obtainable is 640 × 256. Most users opt for a domestic television set, but a dedicated monitor is recommended to get the very best results from the Model B's graphics. Commands exist to draw lines, circles and construct a variety of images on the screen.

Having the power supply unit inside the casing makes the physical design very neat and self-contained, but the interfaces on the back and underside of the case are more numerous than on most machines. This has meant that a large number of devices suitable for expanding the standard computer are available, including several makes of disk drive other than that manufactured specially for the BBC.

In addition to interfaces for disk drive, printer, and an analogue device such as a piece of laboratory equipment or measurement device, there is a facility for networking. This is ideal for classroom use because several users can share one disk drive or printer.

Finally there is 'The Tube' — a sophisticated interface for connecting an alternative microprocessor, either to achieve faster computing, or to run software written for other machines. However, few users seem to have taken advantage of this feature.

BBC Model B Keyboard
The keyboard is a strong point of the BBC Model B, in terms of layout, facilities and quality of construction. The keys are properly sculptured, which means that even a touch typist would feel at home.

The arrow keys are for moving the cursor around the screen for editing text or programs.

The top row of 10 red programmable function keys are particularly useful for educational programs as the user simply has to pick the right answer from up to 10 possibilities.

A pleasing feature is the inclusion of three LEDs (Light Emitting Diodes) to indicate whether the cassette motor is running, and whether the Shift Lock and Capitals Lock Keys have been activated

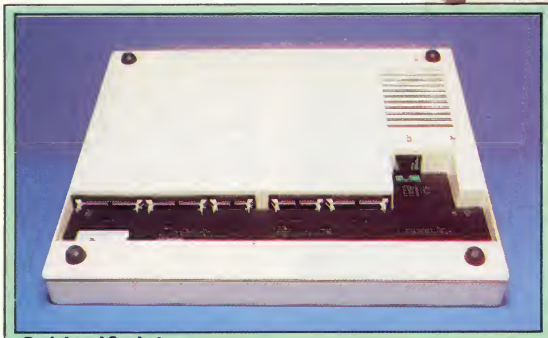
The Disk Drive
The single disk drive for the BBC micro is a very attractive 'extra' but costs around £260 and stores only 100 Kbytes, making it a rather expensive peripheral. However, cheaper drives are available from other sources and should be considered

Foreign Chips
If you look carefully at the chips, you will see that the manufacture of microprocessors is a very international affair. The BBC Model B contains chips made in Malaysia, Japan, Portugal, Scotland, and the USA

Analogue Input
This enables the computer to read a voltage from a non-digital piece of equipment such as a heat sensing device. It is mostly used in the laboratory or for experimentation work

Video Controller
This chip takes information from the user memory and converts it into a video signal for display

Interface Chip
Versatile interface adaptors such as this MOS Technology 6522 look after the interfacing to external devices. Although they are not processors, these chips are as sophisticated as the microprocessor itself

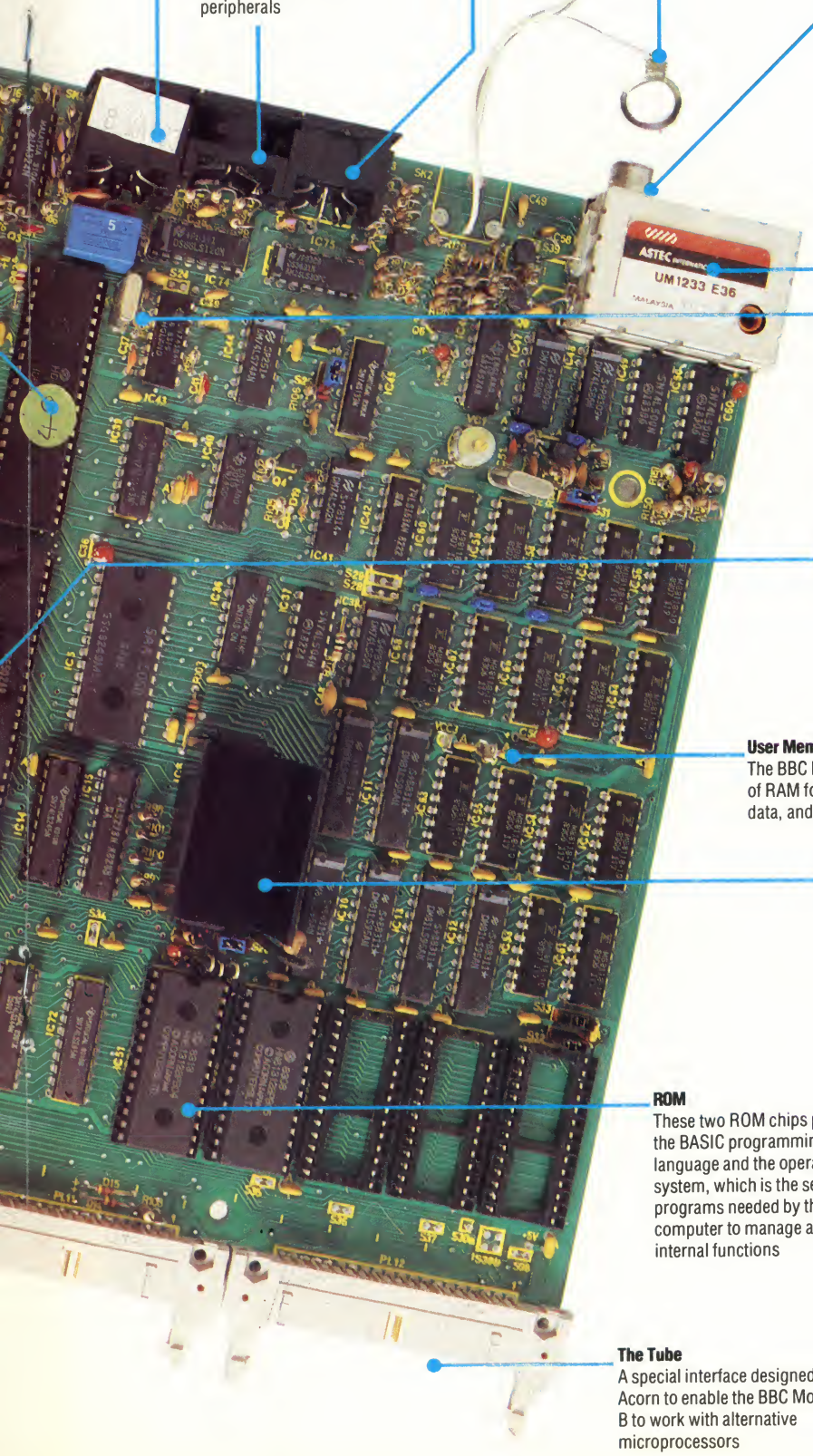


Peripheral Sockets
A view of the underside of the BBC Model B, showing the sockets where the peripherals may be connected

IAN McKINNELL

Printer Port
This is where any printer using a parallel signal, is plugged in

User Port
For experimentation with digital devices and home-built logic circuits



Cassette Interface
Programs can be saved on a domestic cassette recorder in two ways: one giving maximum speed: the other increasing recording reliability

RGB Output
Provides separate signals for the red, green and blue components of the colour video signal. This drives a high-quality colour monitor

Television Output
This is joined to the aerial socket on a television set

Video Output
For use with a monochrome monitor

RS232 Port
A high speed serial interface for use with a number of peripherals

Modulator
This takes the colour signal from the video controller and converts it to an output suitable for a television set

Quartz Crystal
A pulsating crystal that forms the heart of the clock, synchronising all operations

Microprocessor
The MOS Technology 6502 carries out all the processing

User Memory
The BBC Model B contains 32K of RAM for storing programs, data, and graphic displays

ULA
This specially-designed Uncommitted Logic Array does the work of dozens of individual components found in other computers. The piece of metal mounted on top acts as a heatsink to prevent the chip from overheating

ROM
These two ROM chips provide the BASIC programming language and the operating system, which is the set of programs needed by the computer to manage all its internal functions

The Tube
A special interface designed by Acorn to enable the BBC Model B to work with alternative microprocessors

BBC MODEL B

PRICE
£399

SIZE
409 × 358 × 78mm

WEIGHT
3700g

CPU
6502A

CLOCK SPEED
2 MHz

MEMORY
32 Kbytes of RAM
32 Kbytes of ROM including BASIC and sophisticated operating system

VIDEO DISPLAY
8 different graphics modes give a wide choice of displays. Largest text area: 32 lines of 80 characters. Highest resolution graphics: 640×256 pixels. Up to 16 colours at a lower resolution

INTERFACES
Television, monochrome and colour monitors, disk, printer, analogue input, user port, the Tube (for connecting additional microprocessors)

LANGUAGE SUPPLIED
BASIC

OTHER LANGUAGES AVAILABLE
LISP, FORTH, LOGO

COMES WITH
Leads for cassette deck and television. User Guide. 'Welcome' demonstration cassette and brochure

KEYBOARD
Typewriter-style with 74 moving keys, including 10 programmable function keys

DOCUMENTATION
The BBC Microcomputer User Guide shows all the signs of being written by highly trained minds, who seem to assume that their readers have already mastered computers.

Several large chapters are devoted to specialised usage of the system programs, which control the sophisticated graphics, sound and Input/Output features of the machine.

A detailed and very complete explanation of the operation and programming of the 6502 microprocessor is included, and unlike many such sections in other manuals, is not merely a copy of the original Rockwell documentation

Gates And Adders

Binary numbers, 1s and 0s, can be added together using the simple logic of AND, OR, and NOT

We have seen in a previous article (see page 68) how relatively simple transistor circuits can be used to make logical decisions such as AND, OR and NOT. The surprising thing is that these same 'logic gates' are also the building blocks used to perform arithmetic inside the computer. In logic, the inputs to the gates are either zero volts, to represent 'false', or a positive voltage, to represent 'true'. The absence of voltage is usually symbolised by a zero (0) and a positive voltage is usually symbolised by a one (1). When logic gates are used to perform arithmetic, the same zeros and ones are used, but this time they literally represent the ones and zeros being added.

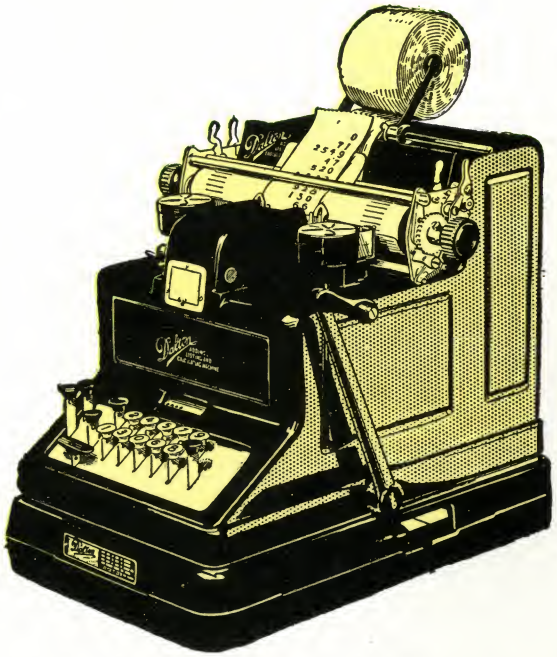
If we want to add two binary digits, there will be only two inputs to the adding circuit, and there can be only four combinations of input — 0 + 0, 0 + 1, 1 + 0 and 1 + 1. From our studies of binary arithmetic, we have learnt that 0 plus 0 equals 0 (as in decimal arithmetic). We also know that 0 plus 1 (or 1 plus 0) equals 1 (again as in decimal arithmetic). The difference from the arithmetic we learnt in school is that in binary, 1 plus 1 equals 0 carry 1. Showing these four additions arithmetically, they would look like this:

X		Y		Z
0	+	0	=	0
0	+	1	=	1
1	+	0	=	1
1	+	1	=	10

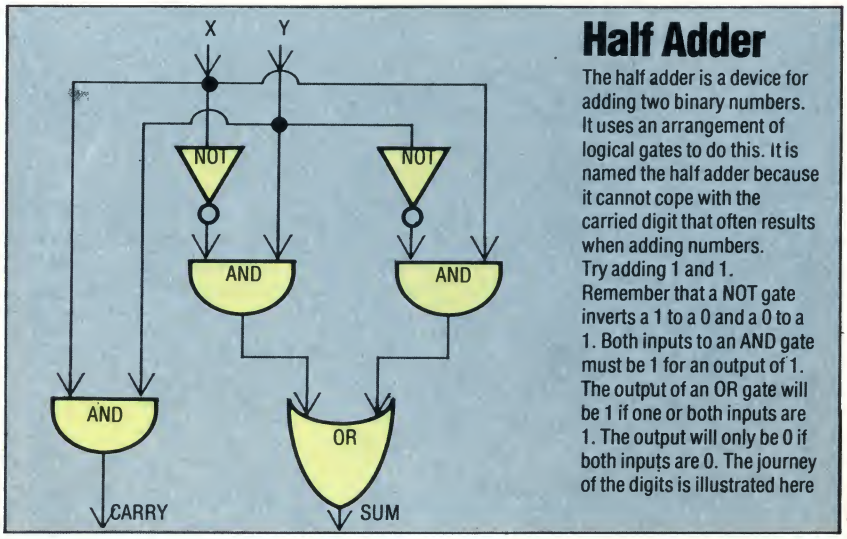
If we were to use an OR gate to do the addition, we would get a false output (0) if both inputs were false (0), and a true output (1) if either of the inputs were true (0 + 1 or 1 + 0).

So far, using a simple OR gate would seem perfectly adequate for adding two binary digits. But wait. If both the inputs are true, the output of a simple OR gate would also be true, but that would be the wrong answer in binary arithmetic. The answer should be a 0 and a carry 1. A simple OR gate would get it right for three out of the four possible input combinations, but three right out of four isn't good enough.

What is needed is a circuit that will give an answer of 0 if both inputs are 0, and an answer of 1 if either of the inputs is 1 and the other is 0, and an answer of 0 if both inputs are 1 (as in the truth table above). This is not as hard as it may seem. If we have two AND gates, with the two inputs going to both gates, but with one input being inverted through a NOT gate to one of the AND gates, and if the other input is being inverted through another NOT gate going to the other AND gate (see illustration), we have a situation where a 0 on both inputs will give a false output from both AND gates, and a 1 on both inputs will similarly give a false output from both gates. On the other hand, a 0 on one input and a 1 on the other will give two true inputs to one of the AND gates. One of these gates will therefore produce a true output. If the two AND gates have their outputs connected to an OR gate, the output of the OR gate will be true only if one, and one only, of the two inputs is true.

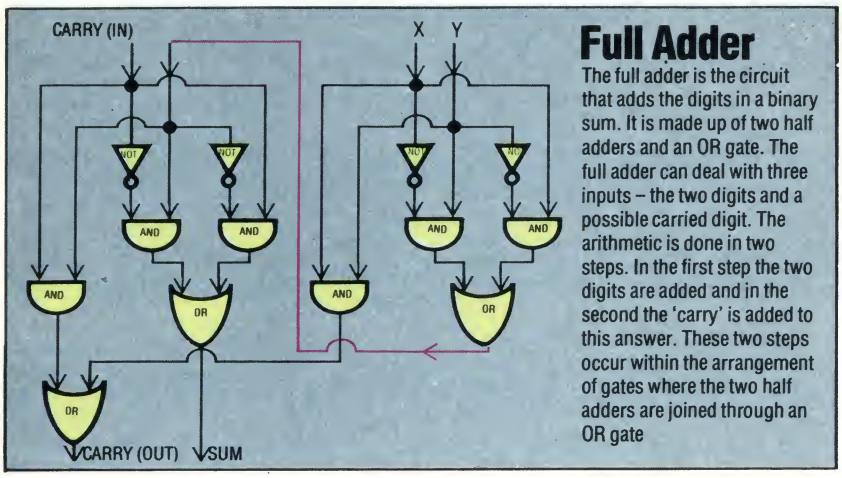


The Desk-Top Adder
Until the recent invention of the electronic calculator, the mechanical adding machine (or cash register) was a common feature in shops and offices. With the exception of a few refinements, it has remained essentially unchanged for 300 years, working through an arrangement of toothed wheels and cogs. The commercial potential for a calculator was quick to be seen. Pascal invented the first adding machine for use in his father's tax office. With division and multiplication developed by Leibnitz, the calculator was ready for business



Half Adder
The half adder is a device for adding two binary numbers. It uses an arrangement of logical gates to do this. It is named the half adder because it cannot cope with the carried digit that often results when adding numbers. Try adding 1 and 1. Remember that a NOT gate inverts a 1 to a 0 and a 0 to a 1. Both inputs to an AND gate must be 1 for an output of 1. The output of an OR gate will be 1 if one or both inputs are 1. The output will only be 0 if both inputs are 0. The journey of the digits is illustrated here

MARK WATKINSON



MARK WATKINSON

Full Adder

The full adder is the circuit that adds the digits in a binary sum. It is made up of two half adders and an OR gate. The full adder can deal with three inputs – the two digits and a possible carried digit. The arithmetic is done in two steps. In the first step the two digits are added and in the second the 'carry' is added to this answer. These two steps occur within the arrangement of gates where the two half adders are joined through an OR gate

This circuit is almost there. The only thing lacking is that an input of two 1s, although giving a 'sum' of 0, correctly, fails to produce a carry signal. However, an additional AND gate, wired in parallel to the two inputs, produces a carry signal when, and only when, both inputs are true. The truth table for the circuit in the illustration, called a half adder, is as follows:

X (input one)	Y (input two)	C ('carry' output)	S ('sum' output)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

It is called a 'half adder' because, in a sense, it is only half adequate. If all we wanted to do was to add a single column of two binary digits, it would be fine. Usually, however, we will want to add two bytes of data together, and each byte contains eight bits. The adder looking after the rightmost column of binary digits would indeed need to be nothing more than a half adder. However, all the adders to the left of that need to be able to accept three inputs — the two digits from 'their' column and any carry from the next column to the right. Consider this addition:

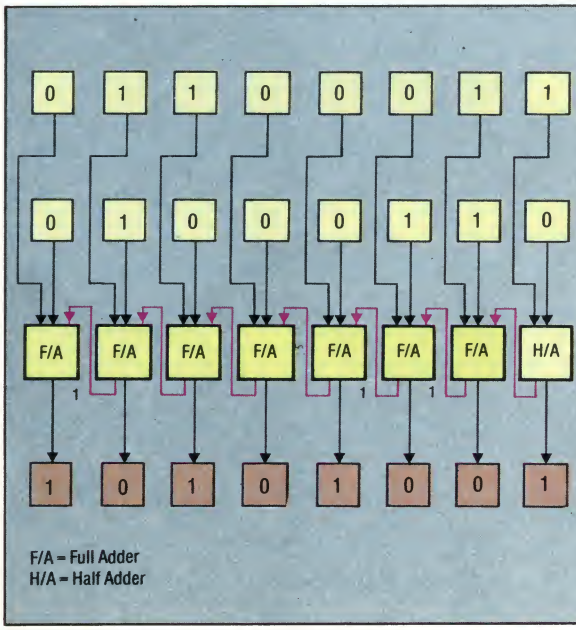
```
  011
+111
----
 1010
```

When adding the 'ones' column, we say 1 and 1 is 0, carry 1' and write a 0 under the 'ones' column. When we add the 'twos' column, we say 1 and 1 is 0, carry 1, plus the carry from the 'ones', is 1, carry 1. We write a 1 under the 'twos' column and carry 1 to the 'fours' column. Here we say 0 and 1 is 1, plus the carry from the 'twos' column, is 0, carry 1. We write the 0 in the 'fours' column and carry 1, which we write under the 'eights' column. In other words, the truth table for a 'full adder' able to handle carries as well as two binary digits, would look like:

X	Y	C(in)	C(out)	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

A full adder can be made using two half adders and an additional OR gate. The 'carry out' of each full adder is connected directly to the 'carry in' of the adder on its left and as many full adders can be chained together in this way as required.

In modern microcomputers, most additions and other arithmetical operations are carried out in large numbers of adder circuits conceptually identical to the ones we have described above. For the most part, though, these adder circuits are included in, and form just a part of, the circuitry of the CPU (the Central Processing Unit). Before the days of large scale integration that culminated in the microprocessor, simpler integrated circuits containing just a few gates were in common use. These circuits are usually called TTL chips (TTL stands for transistor-transistor logic because of the way most of the logic switching is performed by transistors directly coupled together). The inside of a typical CPU consists of a single silicon chip incorporating small areas of RAM and ROM memory, very large numbers of switching circuits and a part known as the ALU or Arithmetic and Logic Unit. The ALU is the part of the CPU containing all the logic gates and adders needed for the computer to perform computations and make logical decisions.



8-Bit Adder

When you add large numbers carefully together the addition is done digit by digit. The computer does the same. In the illustration, two 8-digit binary numbers are being added. The top row contains the first number and the second row the other. Working from the right-hand side, there can be no 'carry' to add to the sum of the first two digits, so a half adder is sufficient. But for the second and subsequent pairs of digits, full adders must be used (any 'carry' is indicated by a 1 below the adder). Try working through the process as the computer does and check your answer in the bottom row

On Record

A programmer's nightmare is losing the masterpiece he has created on screen. With the use of cassettes, the problem is solved

Loudspeaker
The speaker in most cassette recorders is disconnected if the unit is connected to a computer or a hi-fi system

Erase Head
Removes any signal previously recorded on the tape when in record mode

Tape Counter
An essential 'extra' if several programs are going to be stored on each tape

Record/Playback Head
This dual-purpose head records the audio signal onto the magnetic tape and replays it

The Hobbit
Ikon's Hobbit is a dedicated cassette player – one that is designed solely for storing computer programs. It is superior to a music cassette player as the Hobbit is completely under software control. No need to use wind, rewind, play or record, the Hobbit does all that for you. If you want to LOAD a program you type in the name, and the Hobbit searches its catalogue to find the program and its position. It then races through the tape to the right location

IAN McKINNELL

Capstan
A precision-made spindle that rotates at a carefully controlled speed in order to move 1 7/8" of tape a second past the Record/Playback head

Volume
This sets the replay volume and needs to be set carefully or cassettes may not LOAD correctly

When you type a program into your home computer, whether from a book, magazine or your own idea, the information is stored in the computer's RAM. As long as the power is turned on, the program should be quite safe but the instant you turn the computer off it all disappears. This can be something of an irritation; an entire programming session has now disappeared for ever! The next time you want to use that program you must type it all in again. To overcome this problem the makers of home computers generally incorporate a method by which the contents of the computer's memory can be stored in a more permanent form.

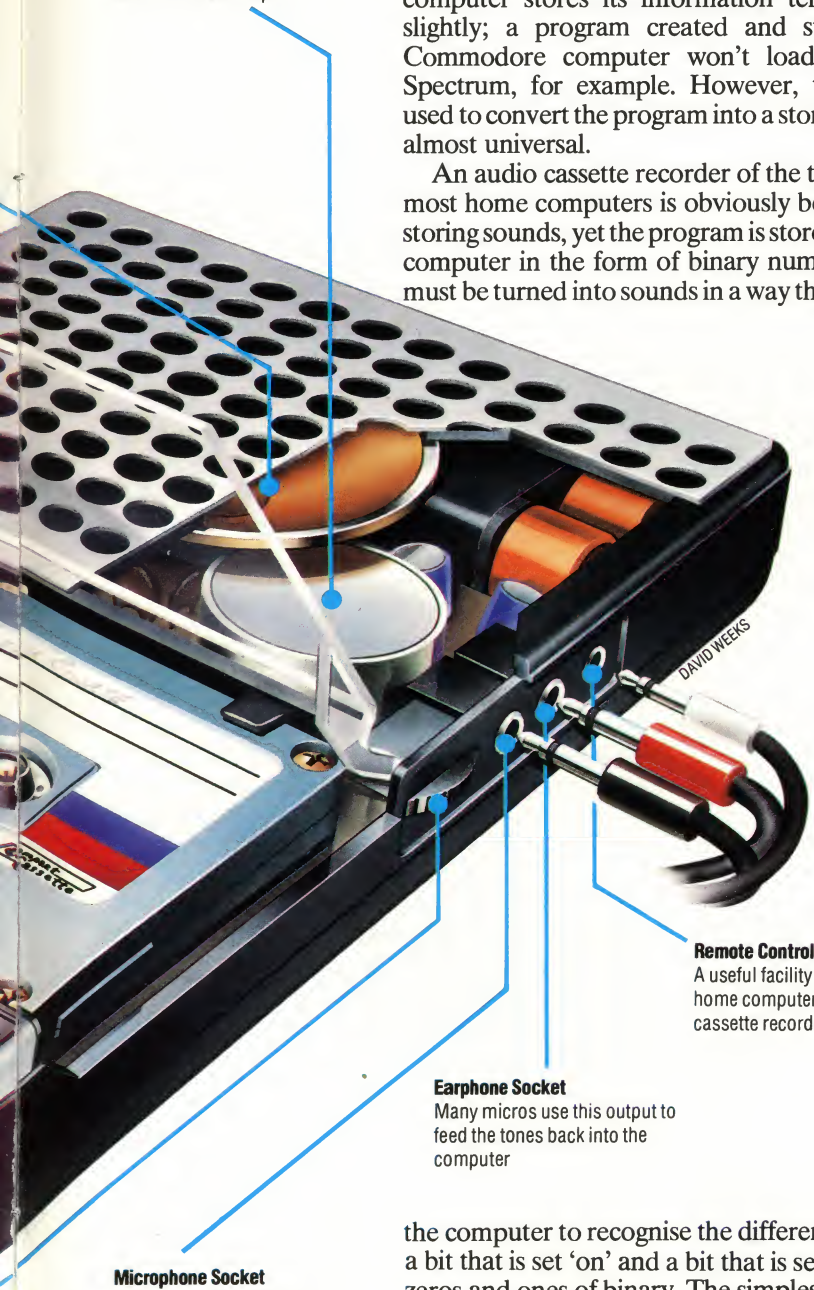
The most common method of storing the program is a cassette tape. Originally chosen because it was both readily available and inexpensive, the system is now used on almost every home computer sold. The way in which each computer stores its information tends to vary slightly; a program created and stored on a Commodore computer won't load on a ZX Spectrum, for example. However, the method used to convert the program into a storable form is almost universal.

An audio cassette recorder of the type used by most home computers is obviously best suited to storing sounds, yet the program is stored inside the computer in the form of binary numbers. These must be turned into sounds in a way that will allow the computer to recognise the difference between a bit that is set 'on' and a bit that is set 'off' — the zeros and ones of binary. The simplest method of doing this is to create one sound that represents a 1 and another that represents a 0. Typically, these are chosen to be a tone of 2,400 cycles for 1 and a tone of 1,200 cycles for 0.

When the command SAVE is typed into the computer the first thing to be recorded on the tape will be a number of seconds of a constant tone. This is done so that when that tape is being played back into the computer at a later date it can tell the difference between the blank tape and the section that holds the program. The first real information to be recorded is the series of tones that represents the characters of the name that we have given to the program. Each character consists of one byte — a total of eight bits — so each character needs eight tones to represent it. However, in order to indicate the beginning and end of each byte the computer usually puts one extra tone at each end. These are called the start and stop bits and their value is always the same: either 1 or 0 depending on the particular computer.

The program itself is stored in much the same way, except that it is often broken down into segments. Typically these are 256 bytes long and they will often include extra information which enables the computer to be sure that it is reloading the correct information. The system used here is quite simple and is called a 'checksum'. The first byte of the segment contains the number of bytes that are held in the segment and the last byte contains a specially calculated number representing the total of all the bytes added together. When the computer reads the cassette back it checks the figures found on the tape with those it has calculated for itself and, if they don't match, informs the user of the mistake.

Certain cassette systems, like the one found on the BBC Micro, extend this checking to the extent of naming and numbering every segment. If an error occurs it is possible to simply wind the tape back a few inches and try again. Other systems, in strong contrast, don't even show the name of the program that is being loaded.



Motor
The motor drives the capstan at a constant speed and also turns the 'supply' and 'takeup' reels to wind and rewind tape

Remote Control
A useful facility that allows home computers to control the cassette recorder

Earphone Socket
Many micros use this output to feed the tones back into the computer

Microphone Socket
Often used as the input for computer data to the recorder. However, it should only be used if the Auxiliary and DIN sockets are not fitted. Careful adjustment of the tone and volume controls are usually needed when this input is used

The Baud Rate

The speed at which the tones are produced and recorded on the tape is usually (and incorrectly) referred to as the Baud rate. The name originates from the Baudot code used in the earliest forms of the electric telegraph and actually relates to the number of times the signal changes per second. A more accurate measure would be the number of bits that are recorded per second. The faster the quoted speed – they range from 300 to 1,200 bits per second – the quicker your programs will be stored on the tape and the less time it will take to load them back into the computer. Unfortunately, the reliability of the system suffers the faster the tones are stored; a speed of 1,200 bits per second is both reliable and sufficiently fast to prevent frustration. Some systems offer two speeds, usually an ultra-reliable slow speed of 300 bits per second and a fast speed of either 1,200 or 2,400 bits per second. Copies of valuable programs can be held in both forms in case of accident.

The cassette tape itself should be of good quality: there is nothing wrong with using audio tape rather than the specially packaged cassettes, but care should be taken to choose a reputable brand and length in excess of C-60 should be avoided. The approximate capacity of a given length of tape can be established by dividing the speed of the cassette interface by 10. This gives the number of bytes that will be stored on the tape each second; a C-60 with 30 minutes on each side where the interface works at 1,200 bits per second, could hold some 432 Kbytes of program.

Safely Stored

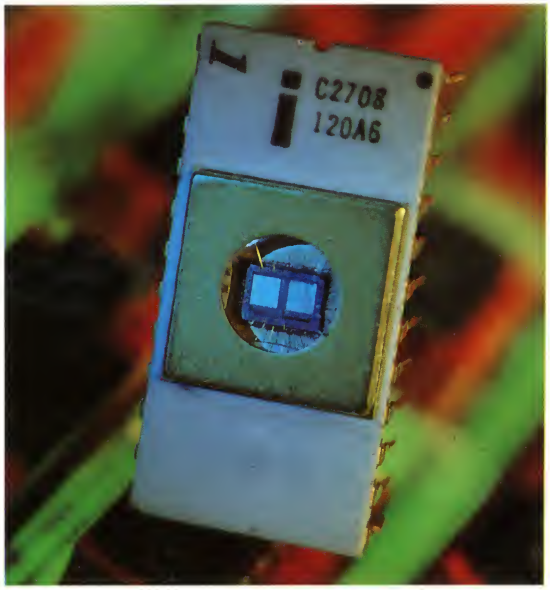
The computer can store thousands of bytes of information in its memory and remember where each one is stored

One way of describing computer memory is in terms of long-term and short-term storage. The long-term type does not lose the information stored and can retain it for long periods even when the power is switched off. Magnetic tape and floppy disks come in this category.

Computers also need fast short-term memory for the temporary storage of programs and results.

Another way of looking at computer memory is to think of it as being either internal or external memory. The internal memory is located inside the computer and is usually fully 'electronic' while external memory is peripheral to or outside the computer. External memory is usually partly mechanical, involving mechanisms such as cassette decks, floppy disk drives or even paper tape punches and readers.

The internal electronic memory is usually called the main memory, while the external memory is referred to as secondary memory or backup memory. These days, internal memory comes in two main varieties – RAM and ROM.



EPROM
The problem with ordinary ROMs is that the memory contents are 'built in' at the manufacturing stage and cannot be changed. EPROMs (Erasable Programmable Read Only Memory) are considerably more flexible. Once programmed, they can be reprogrammed by first erasing the contents and then 'writing' in a new program. EPROMs incorporate a 'window' of silica that allows ultra violet rays to pass through to the interior, causing the capacitors that store the bits in the EPROM to discharge. In the absence of ultra violet light, the capacitors retain their charge indefinitely and the memory contents are retained

Both RAM and ROM are completely electronic devices fabricated in the form of silicon chips and packaged in rectangular plastic cases with sets of parallel tin or silver plated leads. There are further similarities in the way they are selected and 'addressed' by the computer's CPU, but we shall come to that shortly.

The chief functional difference is that ROM memory chips are used to store programs in a permanent form. The pattern of ones and zeros in each memory location is fixed and set at the time of manufacture and cannot be subsequently changed. ROMs are the 'reference libraries' of the computer world. The computer can refer to the contents of the ROM, but is not able to 'write' anything there.

ROM stands for Read Only Memory, read being the term used to describe what the computer does when it 'accesses' or retrieves information from memory. ROMs come in a number of slightly different types, some of which can have the internal program specially removed or erased and can then be re-programmed. A reasonably typical ROM, however, is the 2364 from Intel. This chip is described as being a 65,536 bit ROM, organised as 8 Kbytes of 8 bits. What this means is that the 64 Kbits are grouped together into 8-bit bytes and each 'addressable' location accesses or reads one whole byte. In mathematics $1K = 2^{10}$ (two to the power of ten) or 1,024 so $64K = 64 \times 1{,}024$ or 65,536.

The computer therefore has to be able to select any one of 8,192 (8K) address locations. A close look at the specifications for the 2364 chip reveals that it has 28 pins with one reserved for the +5 volt power supply and one for the ground (earth) connection. This leaves a total of 26 pins. Each byte contains eight bits, so when a byte is read from the chip, the eight bits in that byte have to be conveyed by wires from the chip to the CPU. Consequently, there are eight wires to convey the bits in the byte being read to the CPU. These wires are called the 'data bus'. Eight of the pins on the chip are dedicated to this, one for each bit in the byte.

This leaves 18 pins. One pin is not needed and is not connected. It is retained because it is easier to manufacture chips with an even number of pins. Four pins are used for 'selecting' the chip in various ways. These are the 'output enable' pin, the 'chip enable' pin and the two 'chip select' pins. These pins take signals from the computer to enable the chip to know when it is required.

The remaining 13 pins are the 'address' pins. Each pin is connected to an 'address bus' wire and these carry the address of the byte required, coded in binary form. Thirteen binary digits can give 2^{13} or 8,192 unique combinations of one and zero, so the 13 address lines are just enough to select uniquely each and every of the 8,192 bytes stored in the ROM.

RAMs are the blackboards of the computer world. Programs and data can be stored in them

ROM and RAM

Long-Term Memory
The ROM (Read Only Memory) is analagous to a book in that it is a place where information is stored permanently. You cannot change or remove the data any more than you can alter the words on a printed page

Short-Term Memory
The RAM (Random Access Memory) is more like a filing system than a book, since the information can be changed and the data is not permanent — the RAM is wiped clean when the computer is turned off

temporarily, while the computer is working, and results and other data can also be 'written' there temporarily. RAM memory is generally more complex internally than ROM because every bit in every byte of RAM must be capable of being changed if it is 'written to'. A fairly typical RAM chip is the Intel 2114. Each 2114 RAM chip holds 4,096 bits of memory, and these are organised as 1024 'nibbles' (half bytes) of four bits. This means that each location address will output four bits of data. Two of these chips will therefore be needed to produce a whole Kbyte of data. Each 2114 chip has just 18 pins, two of which are used for the ground and power supply. Four are used for the input/output data lines. One is used for the chip select signal (the signal that tells the chip when it is required or 'selected') and one is used to tell the chip, once it has been selected, if it is being written to or read from. The remaining ten pins are used for the address bus. Ten address lines can uniquely identify 2^{10} locations, or 1,024. If a computer were supplied with 64 Kbytes of RAM, and if Intel 2114 chips were used, a total of 128 RAM chips would be required as two chips are needed for every whole byte. These days it is more usual to use higher density RAM chips that pack more memory into the same space. Using more modern RAM chips, such as the 4164, it is possible to get 64 Kbytes of RAM with just eight chips.

RAM and ROM chips are becoming cheaper and more compact year by year and it is now possible to get 128 Kbits on a single chip. Progress in packing even higher densities into single chips is slowing down, however. The circuitry on the silicon is becoming so minute that the optical techniques used to 'etch' the circuits are barely up to the job. The 'high density' memory chips of the future are likely to be manufactured using electron beam or X-ray etching methods.

Broadly, there are two types of RAM memory in use, known as static and dynamic RAM. There are advantages and disadvantages to both types, but dynamic RAM is now used more commonly than static. Both types lose the memory contents as soon as the power supply is switched off, but dynamic memory needs to have the contents 'refreshed' every few milliseconds. Every bit in memory needs to be refreshed or rewritten without slowing down the CPU's ability to access data here. This means that special and very critical timing circuitry has to be designed, making the circuit designer's job more difficult.

Dynamic memory offers two distinct advantages over static memory. Dynamic memory requires only one transistor per bit, compared with the three transistors normally required for each bit in static memory. This allows more memory to be packed into smaller chips. Most dynamic RAM chips have only 16 pins. The other advantage of dynamic RAMs is that they use less power than their static counterparts. They therefore generate less heat and need smaller, cheaper power supplies.

The advantage of static RAM lies in the simplicity of circuit design. Once the contents of memory have been written, they stay in memory without needing to be refreshed. Each one-bit memory cell requires three transistors, so it is difficult to achieve the high densities that dynamic RAM allows. Static RAM also consumes more power and the extra heat generated complicates the computer's cooling system and may require the use of a cooling fan, making the design a lot more expensive.

Christmas In Basic

We introduce new commands for dealing with data and write a program to work out the number of days until Christmas

This program revises all the topics covered so far in our programming course, and also introduces several new and powerful BASIC statements. The purpose of the program is to calculate the number of days remaining until Christmas.

If you look at the program listing, you will see that it starts with a list of the variables used. This practice is certainly not essential, but is advisable as it can make your programs much easier to understand when you come to look at them later. Some versions of BASIC allow variables to have long names, DAY for example, rather than the single letters we have been using. If you are lucky enough to have a BASIC that allows long variable names, choose meaningful names. DAY, MONTH or DAYNUM are much better than A, X or D. If you have no choice in the matter because your BASIC does not allow long variable names, listing the variables at the top of the program makes it almost as 'readable'.

When the program is run, the first thing that will appear on the screen will be the PRINT statements starting at line 230. These state briefly what the program will do and then prompt the user to type in the date in the form shown, using commas to separate the day, month and year.

The first unfamiliar statement will be in line 300. This is a DIMension statement. It is used to set the number of items or elements allowed in the array labelled X. An array, sometimes called a subscripted variable, is like an ordinary variable except that the box contains several compartments. In line 300 we are creating a variable called X with 13 compartments inside the box. We shall return to the subject of arrays and the DIM statement in more detail later in the course.

```
310 INPUT D, M$, Y
```

This line is an ordinary INPUT statement except that it expects three inputs. D is a numeric variable that will contain today's date. Y is another numeric variable for the year. M$ is slightly different. It is called a 'string variable' and this is indicated by the $ (dollar) sign. A string variable accepts characters from the keyboard as well as numbers. If, for example, we type 23, JANUARY, 1983, variable D will be assigned the value 23, variable M$ will be assigned the character string JANUARY and variable Y will be assigned the value 1983.

```
330 GOSUB 560 REM 'NO OF MONTH' ROUTINE
```

This statement instructs the program to branch to the subroutine starting at line 560. Note, also that a REMark has been inserted on the same line. If there is room on the line, it is not always necessary to put REMs on a new line. This particular subroutine is only used by the main program once, and strictly speaking could just as easily have been incorporated into the main program. Making it into a subroutine just keeps this part separate from the rest of the program.

When the program was originally written, a number was used for the month and this part of the program was not needed. Later it was decided to allow the month to be entered as a typed word spelled out in full. In order to convert the spell-out month into its equivalent number, the extra program now forming this subroutine was written separately. The only change needed to the main (original) program was to add a single GOSUB statement. This subroutine illustrates the ease with which programs can be built up in blocks and linked together using the GOSUB and RETURN statements.

The subroutine itself is very simple, but illustrates how clever BASIC is at manipulating character strings. Suppose we had entered JANUARY as the month part of the INPUT statement. Variable M$ would then be assigned the character string JANUARY. The first line of the subroutine is:

```
560 IF M$ = "JANUARY" THEN LET M = 1
```

This statement compares the contents of M$ with the characters inside the double quotation marks. If they are the same (as they are in this case) the line goes on to set the value of numeric variable M to 1. Do not confuse variable M with variable M$. They are different. Only one can contain a string variable, the one with the $ sign! After checking to see if M$ is the same as the string JANUARY, the program moves to the next line and checks to see if the contents of M$ are the same as FEBRUARY. It is not, so M is not set to 2. Only where the match is correct will variable M be set to a value, and that value is the same as the number of the month — 1 for January, 3 for March and so on.

On getting to line 680 BASIC RETURNs to the main program, to the line after the GOSUB statement. This is line 340. It contains a REM but no comment. It is inserted simply to space out the program and to make it easier to read.

Lines 350 to 370 are a FOR—NEXT loop. This increments the value of I, starting with 1 and counting up to 13. The variable I is used as the

subscript of the array X in line 360. It should be examined carefully.

```
360 READ X (I)
```

READ is a new statement we have not encountered before. READ is always used with a corresponding DATA statement. The DATA statement for this line is in line 510:

```
DATA 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 25, 0
```

These numbers, except for the last two, are the numbers of days in each month of the year. The two lines are equivalent to 13 separate LET statements

```
LET X(1) = 31
LET X(2) = 28
LET X(3) = 31
LET X(4) = 30
LET X(5) = 00
LET X(6) = 00
LET X(7) = 31
LET X(8) = 31
LET X(9) = 30
LET X(10) = 31
LET X(11) = 30
LET X(12) = 25
LETX(13)=0
```

The loop set up in line 350 makes I count up from 1 to 13 so we were able to substitute X(I) for X(1), X(2), X(3) etc.

Before returning to this program, let's consider a far simpler small program:

```
10 READ A, B, C
20 LET D = A + B + C
30 PRINT D
40 DATA 5, 10, 20
```

Here, the READ statement in line 10 reads the first item of DATA in line 40 and 'writes' its value into the first variable. In other words, it assigns the value 5 to variable A. READ then reads the next item of data and puts it in the next variable. This program makes A = 5, B = 10 and C = 20. It then adds these and assigns the result to variable D. This result, 35, is then PRINTed in line 30.

Back to the 'Christmas' program. The first time round the loop starting in line 350, the value of I is set out to 1. Line 360 is therefore equivalent to READ X(1). The corresponding data item in line 510 is 31 (the first item). Consequently X(1) is set to 31.

The second time round the loop, I becomes 2 so line 360 is equivalent to READ X(2). The next data item in the DATA line is 28. This means that X(2) is set to 28. In this way all 13 'compartments' in the subscripted variable X are filled up with the number of days in each month; except for the 12th compartment, which has only 25 days in it, and the 13th, which has 0. (Can you see why?)

```
390 GOSUB 750 REM 'LEAP YEAR' ROUTINE
```

This line directs the program to a subroutine that checks if the year entered is a leap year or not.

```
100 REM  LIST OF VARIABLES
110 REM
120 REM   D = TODAY'S DATE
130 REM   M$ = NAME OF MONTH
140 REM   Y = YEAR
150 REM   I = INDEX 1
160 REM   X = ARRAY FOR DAYS IN EACH MONTH
170 REM   R = REMAINING DAYS
180 REM   M = NO. OF MONTH
190 REM   L = INDEX 2
200 REM   Z = INT. VALUE OF Y/4
210 REM
220 REM
230 PRINT "THIS PROGRAM CALCULATES"
240 PRINT "THE NUMBER OF DAYS REMAINING"
250 PRINT "UNTIL CHRISTMAS"
260 PRINT
270 PRINT "ENTER TODAY'S DAY,MONTH,YEAR"
280 PRINT "E.G. 12,JULY,1984"
290 PRINT
300 DIM X(13)
310 INPUT D,M$,Y
320 REM
330 GOSUB 560  REM 'NO OF MONTH' ROUTINE
340 REM
350 FOR I = 1 TO 13
360 READ X(I)
370 NEXT I
380 REM
390 GOSUB 750  REM  'LEAP YEAR' ROUTINE
400 REM
410 LET R = X(M) - D
420 FOR L = M TO 11
430 LET M = M + 1
440 LET R = R + X(M)
450 NEXT L
460 REM
470 IF R = 1 THEN GOTO 500
480 PRINT "THERE ARE";R;"DAYS LEFT UNTIL CHRISTMAS"
490 GOTO 520
500 PRINT "THERE IS 1 DAY LEFT UNTIL CHRISTMAS"
510 DATA 31,28,31,30,31,30,31,31,30,31,30,25,0
520 END
530 REM
540 REM
550 REM
560 IF M$ = "JANUARY" THEN LET M = 1
570 IF M$ = "FEBRUARY" THEN LET M = 2
580 IF M$ = "MARCH" THEN LET M = 3
590 IF M$ = "APRIL" THEN LET M = 4
600 IF M$ = "MAY" THEN LET M = 5
610 IF M$ = "JUNE" THEN LET M = 6
620 IF M$ = "JULY" THEN LET M = 7
630 IF M$ = "AUGUST" THEN LET M = 8
640 IF M$ = "SEPTEMBER" THEN LET M = 9
650 IF M$ = "OCTOBER" THEN LET M = 10
660 IF M$ = "NOVEMBER" THEN LET M = 11
670 IF M$ = "DECEMBER" THEN LET M = 12
680 RETURN
690 REM
700 REM
710 REM
720 REM NOTE: THIS ROUTINE DOES NOT CHECK
730 REM       FOR LEAP YEARS AT THE END OF
740 REM       EACH CENTURY
750 LET Y = Y / 4
760 LET Z = INT(Y)
770 IF Y - Z = 0 THEN GOTO 790
780 RETURN
790 LET X(2) = X(2) + 1
800 RETURN
```

```
750 LET Y = Y/4
760 LET Z = INT(Y)
770 IF Y-Z = 0 THEN GOTO 790
780 RETURN
790 LET X(2) = X(2) + 1
800 RETURN
```

A leap year is defined as one which is wholly divisible by the number 4. If it is a century, it must also be divisible by 400 to qualify as a leap year. To keep it simple, we have not attempted to check the century, only the divisibility by 4.

Line 750 sets Y to the old value of Y (the year) divided by 4. The new Y will be a whole number if the year is exactly divisible by 4. Otherwise it will have a decimal fraction.

Line 760 uses the function INT to find the 'integer' value of Y. Integer means whole number. While having no effect on integers, the INTeger function will round down fractional numbers to the nearest whole number. The number to be rounded down is placed in brackets after INT. Alternatively, a variable name can be put in the brackets. So LET Z = INT(496.25) would set Z to 496.

Line 770 subtracts Z from Y and checks to see if the result is 0. If it is, it means the year is a leap year (as there was no decimal fraction in the new Y). If that is the case, the program branches to line 790 using GOTO. Line 790 adds 1 to the second item in the array (the second item was 28, the number of days in an ordinary February).

If the result of the subtraction in line 770 was not zero, X(2) is left as it is and the subroutine RETURNs to the main program, to line 400.

Line 400 is another REM used just to space out the program to aid readability. The next line that actually does something is 410, where R is the variable holding the number of remaining days. It is set here to the number of days in the month entered minus the day entered. If we had entered, for example, 12, FEBRUARY, 1983, D would be equal to 12 and M would be 2. Therefore X(M) would be the same as X(2) and the second item in the X array is 28 (it would not have had 1 added to it as 1983 is not a leap year). Consequently R will be set to 28 – 12, i.e. 16, the number of days remaining in the current month, February.

Line 420 starts another loop.This one is designed to increment the value of M. Can you see why we say FOR L = 1 TO 11 rather than FOR L = 1 TO 12? If M was 2 because we had entered the month as FEBRUARY, line 430 will increment it to 3. Line 440 then sets R, the number of days remaining, to the old R plus X(M). The latter is now equivalent to X(3) since M has been incremented by 1. The value of X(3) is 31, the number of days in March. Line 440 therefore sets the new value of R to 16 + 31 (16 was the result of subtracting 12 from 28).The next time round the loop, M is incremented to 4 and the number of days in April, X(4), is added to the old value of R. The variable R therefore becomes 16 + 31 + 30.

The last circuit through the loop occurs when L = 11, and X(12)'s value, 25, is added to R.

What happens to the loop if a December date is input, so that M = 12? Because of the discrepancy in the limits, some machines skip the loop entirely, while others execute it once, so that X(13) is added to R. X(13) has been set to 0 to give the correct result.

```
470 IF R = 1 THEN GOTO 500
```

This line simply checks if there is only one day remaining to Christmas so that we get a grammatically correct sentence on the screen. If R is not 1, there must be more than one day remaining, so the PRINT statement in line 480 will be grammatically correct.

So that's all there is to it. The version of BASIC we have used should run on most computers (see the 'Basic Flavours' box) except possibly for the 'leap year' subroutine. BASIC is very inconsistent in the way it uses LET. If lines like IF M$ = "SEPTEMBER" THEN LET M = 9 do not work on your computer, the subroutine can be rewritten like this:

```
560 IF M$ = "JANUARY" THEN GOTO 900
570 IF M$ = "FEBRUARY" THEN GOTO 910
580 IF M$ = "MARCH" THEN GOTO 920
:
900 LET M = 1
905 RETURN
910 LET M 3 2
915 RETURN
920 LET M = 3
925 RETURN 925
(. . . and so on)
```

This solution is more space-consuming and less easy to follow with all its GOTOs and RETURNs. However, it does demonstrate that there are usually several ways of solving every problem.

Basic Flavours

READ DATA

These commands are not available on the ZX81, so delete lines 300, 350 – 370 and 510. Add:

```
10 DIM X(13)
20 FOR K = 1 TO 13
30 PRINT "INPUT ITEM NO.";K
40 INPUT X(K)
50 NEXT K
60 STOP
```

RUN the program and enter the data. Delete lines 10–60 and SAVE the program, which also saves the contents of the array. After LOADing the program in future, use GOTO 100 rather than RUN, thus preserving variables

INPUT

The ZX81 requires an INPUT command for each item, so add 285 PRINT "WHEN PROMPTED", and:

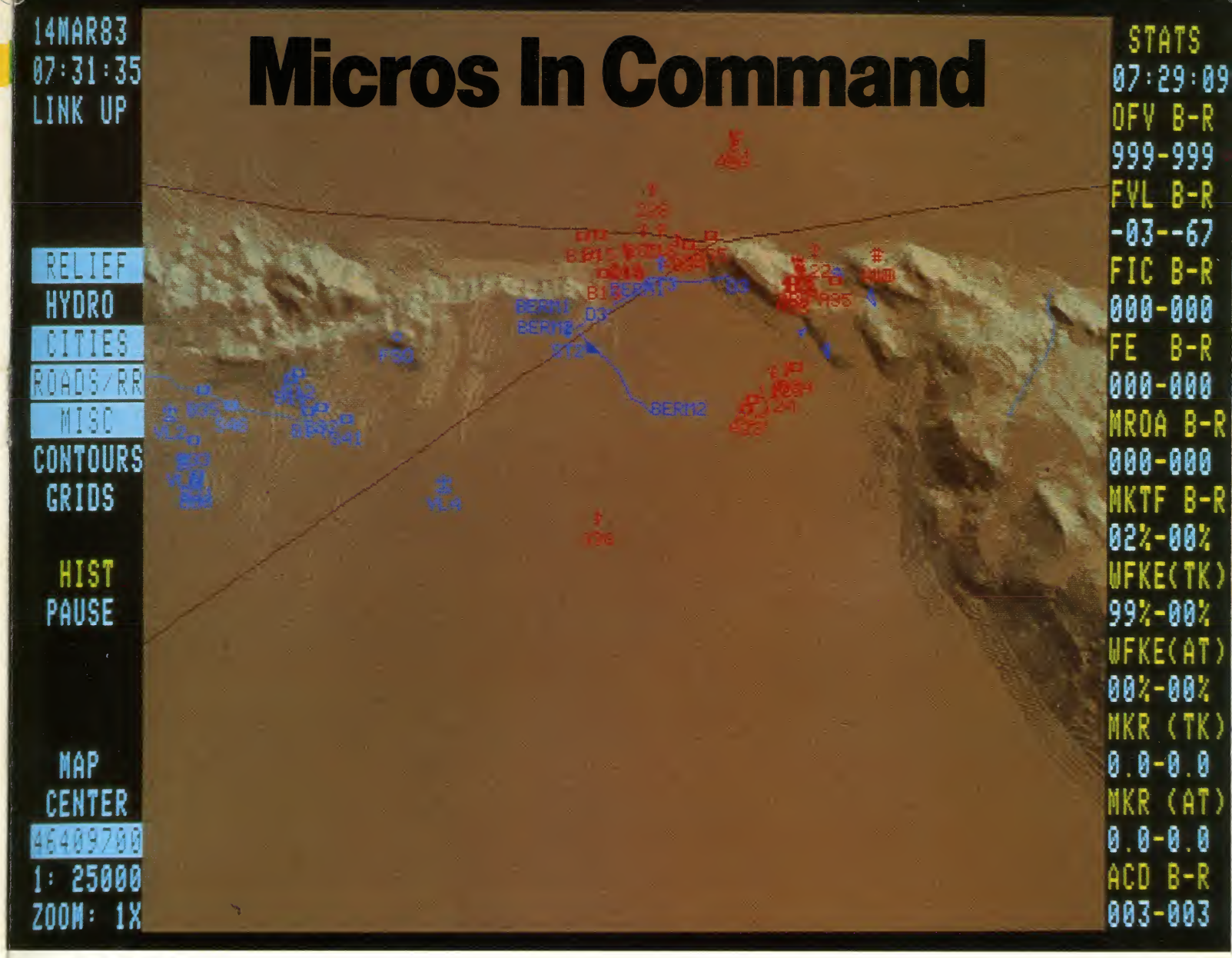
```
310 PRINT "INPUT DAY"
312 INPUT D
```

with similar lines for month and year

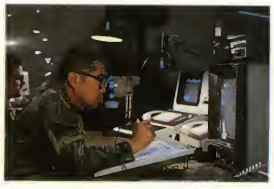
REM

On the BBC Micro, Commodore 64 and Vic-20, REM statements at the end of a program line must be preceded by a colon(:)

Micros In Command



On The Battlefield
The photographs show the US army engaged in mock battle in the Mojave Desert. Machine guns and other weapons are connected to laser systems and their targets fitted with sensory devices. When the laser beam strikes a sensor, the computer records the accuracy of the strike



BRIAN WOLFF

The armed forces of today are probably the most computerised 'industries' in the world. The modern soldier, as well as being skilled in traditional combat and tested for endurance, must also be capable of operating the technology that modern weapons employ.

The Falklands War proved how devastating the new generation of microchip weapons can be. The Exocet missile that sank *HMS Sheffield* was guided by computer, and the pilot did not even have to see the target and take aim in the conventional way.

Computers are proving immensely valuable in tanks where accuracy and speed are essential. British army Chieftain tanks are equipped with a computer that assesses vital factors such as wind direction, barrel wear, the type of ammunition being used and the angle of sight. From this information it calculates the correct tracking and the exact aim of the target. Such a computer can enable a tank to score nine hits out of nine shots, within 53 seconds and at a range of two miles.

A similar computer is being used by other NATO armies. This is a Belgian system that uses a laser range finder, sensors, a computer and an optical sight. The sensors measure factors such as wind and barrel wear and the computer calculates the angles and then displays a set of cross-hairs on the sight. As the cross-hairs follow the target, so does the gun.

The Cruise missile uses several computers to aim its warhead within 60 feet of a target after a flight of nearly 2,000 miles. As the missile is launched, its flight is guided by its computer which stores all the information of the flight path. The computer continually makes fine adjustments to the flight trajectory. As the missile approaches its target, the computer activates its final guidance system. The computer is able to identify the target from a 'picture' composed of millions of numbers in its memory. Once the missile's detectors see the target, they relay a further digital picture to the computer. The computer then guides the missile until both sets of numbers match exactly and then directs it to the target.

The development of computers for use in military technology has been of benefit by accelerating the pace of computer science and design. Without the vast quantity of money spent in the military laboratories, it is quite possible that we would never have seen the £100 home computer in our own lifetimes.

THE HOME COMPUTER COURSE BINDER

Now that your collection of Home Computer Course is growing, it makes sound sense to take advantage of this opportunity to order the two specially designed Home Computer Course binders.

The binders have been commissioned to store all the issues in this 24 part series.

At the end of the course the two volume binder set will prove invaluable in converting your copies of this unique series into a permanent work of reference.

Buy two together and save £1.00

✻ Buy volumes 1 and 2 together for £6.90 (including P&P). Simply fill in the order form and these will be forwarded to you with our invoice.

✻ If you prefer to buy the binders separately please send us your cheque/postal order for £3.95 (including P&P). We will send you volume 1 only. Then you may order volume 2 in the same way – when it suits you!

Overseas readers: This binder offer applies to readers in the UK, Eire and Australia only. Readers in Australia should complete the special loose insert in Issue 1 and see additional binder information on the inside front cover. Readers in New Zealand and South Africa and some other countries can obtain their binders **now**. For details please see inside the front cover.

Binders may be subject to import duty and/or local tax.

THE LAST WORD IN LOGIC